II. Pengenalan HTTP *Kernel*

Pengantar

Sebelum melangkah lebih jauh, kita akan membahas tentang inti dari *framework* Symfony yaitu HTTP *Kernel*. Sebagai informasi, Symfony adalah *framework* PHP pertama yang membuat *class set* untuk mengabstraksi protokol HTTP. *Class set* (*component*) ini dinamakan HTTP *Kernel*.

HTTP *Kernel* adalah inti dari *framework* Symfony itu sendiri maupun *framework* lain yang menggunakan HTTP *Kernel* Symfony seperti Laravel *Framework*, Silex *Framework* maupun Gareng *Framework*. Maksudnya, meskipun implementasinya berbeda, namun secara karakteristik mereka tetaplah sama.

Sebagai informasi, Silex dan Gareng adalah *micro framework* yang dikembangkan menggunakan HTTP *Kernel* sebagai alternatif Symfony dan sekarang sudah ditinggalkan. Ini karena Symfony sendiri sekarang secara *default* adalah *micro framework* sehingga tidak perlu lagi mencari alternatif *framework* yang memiliki *foot print* kecil dan ringan.

Pada pembahasan kali ini, harapan saya teman-teman dapat memahami apa yang terjadi pada *framework* Symfony maupun *framework* lain berbasis HTTP *Kernel* Symfony seperti Laravel.

The Heart of Framework

Seperti yang sudah disinggung di atas, HTTP *Kernel* adalah inti dari *framework* Symfony. Pada aplikasi kita, HTTP *Kernel* diwakili oleh *class* `App\Kernel` yang meng-*extends class* `Symfony\Component\HttpKernel\Kernel`. Instansiasi *class* `App\Kernel` dapat dilihat pada *file* `public/index.php` sebagai berikut:

```php
$kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);
```

Bila diperhatikan, pada *file* `index.php`, *object* `Kernel` hanya memanggil dua *method* yaitu `handle()` dan `terminate()` saja. *Yup* memang sesimpel itulah *framework* Symfony bekerja sebenarnya, namun karena kita belum tahu, kita menganggapnya sangat sulit.

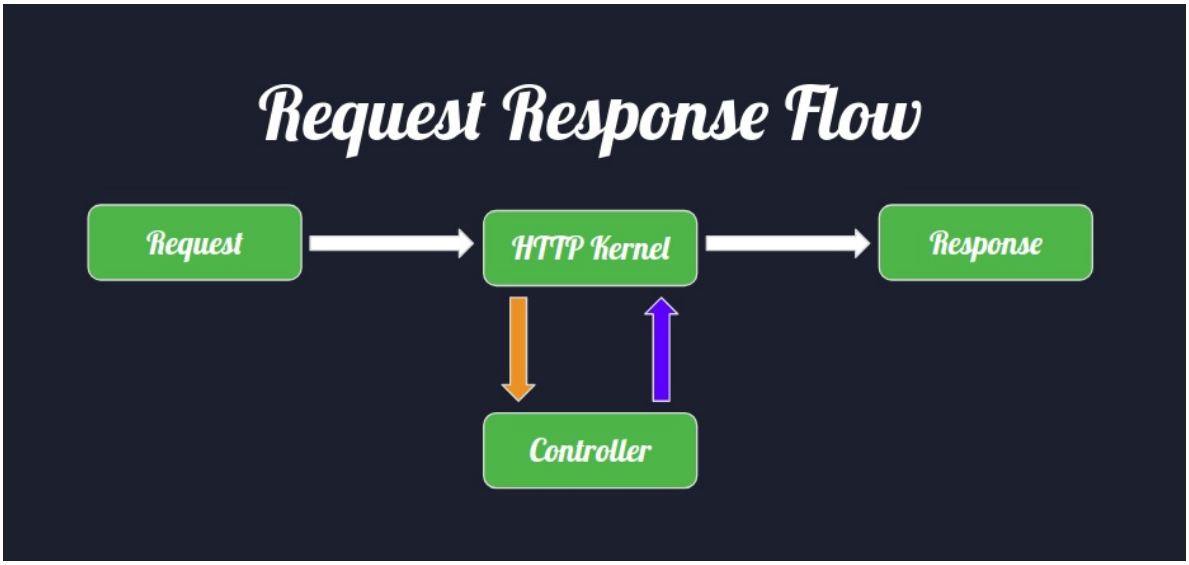
Method `terminate()` sendiri sebenarnya opsional kalau kita mengimplementasikan HTTP *Kernel* sendiri seperti pada *framework* Gareng yang tidak mengimplementasikannya.

Oh iya, dari tadi kita membahas tentang Gareng *Framework* tapi belum tau wujudnya ya? *Repository* dari Gareng *Framework* dapat dilihat pada halaman Github https://github.com/KejawenLab/GarengFramework.

Kembali ke HTTP *Kernel*, tugas utama dari HTTP *Kernel* adalah menerima *request*, memprosesnya lalu mengembalikan *response*. Hanya sebatas dan sesimpel itu tugasnya.

Siklus *Request Response*

Secara sederhana, siklus dari *request* hingga *response* dapat digambarkan sebagai berikut:



Proses di atas dapat dijelaskan sebagai berikut:

> 1. *Setiap request yang dikirim oleh client akan dikonversi menjadi object dari class* `Symfony\Component\HttpFoundation\Request` .
> 2. *Object dari* `Symfony\Component\HttpFoundation\Request` *akan di-passing ke object dari HTTP Kernel melalui method* `handle()` .
> 3. *Method* `handle()` *akan mencari controller yang sesuai untuk request yang dikirim oleh client.*
> 4. *Controller memproses dan mengembalikan object dari* `Symfony\Component\HttpFoundation\Response` .
> 5. *Object HTTP Kernel mengembalikan response ke client.*

Cara Menemukan *Controller*

Pada *framework* berbasis HTTP *Kernel* Symfony, *controller* adalah *callable object* (tipe data `callable` pada PHP) yang bertugas memproses *request* dan mengembalikan *response* kepada HTTP *Kernel*. Karena *controller* adalah *callable object* maka *framework* turunan seperti Laravel, Silex maupun Gareng mendukung simpel *controller* berupa `Closure` sebagai berikut:

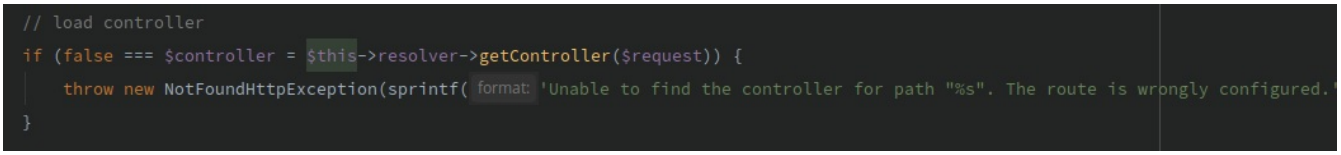
Source: Laravel

Meski bukan *best practice* namun *controller* model seperti itu tetap bisa digunakan dan bisa jadi solusi ketika kita malas untuk mempelajari sebuah *framework* lebih dalam. Sebagai contoh, kita terbiasa menggunakan Symfony, lalu kemudian kita mendapatkan *proyek* membangun aplikasi menggunakan Laravel dengan *deadline* yang ketat. Cara di atas bisa jadi solusi singkat ketimbang harus mempelajari macam-macam tipe *controller* yang ada pada Laravel walaupun sekali lagi bukan *best practice*.

Sayangnya, fitur ini justru tidak tersedia di Symfony *Framework* karena *router* Symfony di-*compile* dan di-*dump* menjadi *optimized object* dan di-*cache* sehingga tidak memungkinkan melakukan *dumping* terhadap *object* `Closure` .

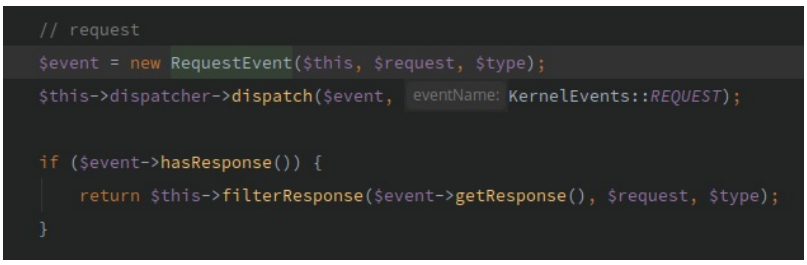
Kembali ke cara Symfony menemukan *controller*, pada HTTP *Kernel* tugas untuk menemukan *controller* diberikan kepada *object* dari `Symfony\Component\HttpFoundation\Request\ControllerResolverInterface` melalui *method* `getController()` .

Method `getController()` ini dipanggil pada *method* `handleRaw()` pada *file* `Symfony\Component\HttpKernel\HttpKernel` seperti pada gambar berikut:



Actual call dari *method* tersebut adalah *object* dari *class* `Symfony\Component\HttpKernel\Controller\ControllerResolver` .

Proses di atas sebenarnya terjadi setelah *route* ditemukan (*route matched*), sedangkan proses menemukan *route*-nya sendiri terjadi pada *event* `RequestEvent` melalui *listener* `Symfony\Component\HttpKernel\EventListener\RouterListener` pada *method* `onKernelRequest()` . *Event* ini di-*trigger* pada HTTP *Kernel* pada baris berikut:



Event System pada HTTP *Kernel*

Seperti yang sudah sedikit disinggung di atas, pada HTTP *Kernel* Symfony terdapat *event* yang di-*trigger* sesuai dengan skopnya masing-masing. *Event-event* ini adalah implementasi dari prinsip *Open/Close Principle* pada *SOLID Principle* dan menerapkan *observer pattern* pada sisi *design pattern*-nya menggunakan *component Event Dispatcher*.

Pada Symfony, terdapat beberapa *built-in event* antara lain:

1. *Kernel Request*

- *Event ini dipanggil sebelum controller ditemukan.*
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\RequestEvent`
- *Dapat digunakan untuk meng-intercept request maupun memodifikasi response.*
- *Contoh penerapan event ini adalah* `RouteListener` *seperti pada pembahasan sebelumnya.*
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.request` *untuk mengetahui listener dari event kernel request*

2. *Kernel Controller*

- *Event ini dipanggil setelah controller ditemukan namun sebelum benar-benar dieksekusi.*
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\ControllerEvent`
- *Dapat digunakan untuk memproses custom annotation*
- *Contoh penerapan event ini adalah* `Symfony\Component\HttpKernel\DataCollector\RequestDataCollector` *yaitu pada method* `onKernelController()`
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.controller` *untuk mengetahui listener dari event kernel controller*

3. *Kernel Controller Arguments*

- *Event ini dipanggil setelah event Kernel Controller dipanggil sebelum controller benar-benar dieksekusi.*
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\ControllerArgumentsEvent`
- *Event ini jarang diimplementasikan secara eksternal.*
- *Berfungsi untuk menemukan argument yang pas serta urutan yang tepat dari method controller seperti mem-passing object* `Request` *atau route argument seperti* `{id}` .
- *Dapat digunakan untuk memanipulasi argument yang di-passing ke controller.*
- *Contoh penggunaan event ini adalah* `Symfony\Component\HttpKernel\EventListener\ErrorListener` *yaitu pada method* `onControllerArguments()` .
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.controller_arguments` *untuk mengetahui listener dari event kernel controller arguments*

4. *Kernel View*

- *Event ini dipanggil setelah controller dieksekusi dan hanya jika controller tidak mengembalikan object* `Response` .
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\ViewEvent` .
- *Dapat digunakan untuk melakukan transformasi dari (misalnya) string HTML ke object* `Response` .
- *Secara default Symfony tidak mengimplementasikan event ini.*
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.view` *untuk mengetahui listener dari event kernel view*

5. *Kernel Response*

- *Event ini dipanggil setelah controller diekseksui atau setelah event* `ViewEvent` *mengembalikan object* `Response` .
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\ResponseEvent` .
- *Dapat digunakan untuk memanipulasi response seperti menambahkan header atau cache pada response.*
- *Contoh penggunaan event ini adalah* `Symfony\Component\HttpKernel\EventListener\ResponseListener` *yaitu pada method* `onKernelResponse()` .
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.response` *untuk mengetahui listener dari event kernel response*

6. *Kernel Finish Request*

- *Event ini dipanggil setelah event* `ResponseEvent` *dipanggil.*
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\FinishRequestEvent` *.*
- *Event ini sangat jarang (malah belum pernah) saya temukan penggunaannya selain pada internal Symfony.*
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.finish_request` *untuk mengetahui listener dari event kernel finish request*

7. *Kernel Terminate*

- *Event ini dipanggil setelah response dikirimkan (mengeksekusi method* `send()` *seperti pada* `index.php` *)*
- *Dipanggil melalui* `$kernel->terminate()` *pada* `index.php`
- *Sama seperti event sebelumnya, event ini juga sangat jarang saya temukan penggunaannya.*
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.terminate` *untuk mengetahui listener dari event kernel terminate*

8. *Kernel Exception*

- *Event ini dipanggil ketika terjadi error pada saat HTTP Kernel meng-handle request.*
- *Class dari event adalah* `Symfony\Component\HttpKernel\Event\ExceptionEvent` *.*
- *Dapat digunakan untuk memodifikasi error message seperti mengubah error dari HTML menjadi JSON.*
- *Contoh penggunaan event ini adalah* `Symfony\Component\HttpKernel\EventListener\RouterListener` *yaitu pada method* `onKernelException()` *.*
- *Jalankan perintah* `php bin/console debug:event-dispatcher kernel.exception` *untuk mengetahui listener dari event kernel exception*

Penutup

Bila membaca penjelasan di atas, ditambah ada banyak *event* yang dipanggil oleh Symfony, mungkin kita berfikir bahwa *framework* Symfony itu lambat, namun nyatanya jika dibandingkan dengan *enterprise framework* yang selevel seperti Zend *Framework* (sekarang Laminas), Symfony masih lebih cepat. Bahkan jika melihat *benchmark* sebelumnya, Symfony masih lebih cepat dari YII maupun Laravel.

Dan karena sifatnya hanya seputar *request* dan *response*, Symfony tidak menyebut dirinya sebagai MVC *framework* namun hanya *request response framework*. Itu berarti, Symfony jauh lebih fleksibel dari sekedar MVC, tapi lebih dari itu, Symfony bisa dapat menerapkan MVC, DDD, Hexagonal bahkan CQRS atau *micro service* sekalipun.